

日 本 国 特 許 庁
JAPAN PATENT OFFICE

J1046 U.S. PTO
10/085455
02/27/02

#5

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出 願 年 月 日

Date of Application:

2001年 2月28日

出 願 番 号

Application Number:

特願2001-055996

出 願 人

Applicant(s):

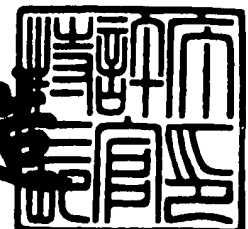
インターナショナル・ビジネス・マシーンズ・コーポレーション

CERTIFIED COPY OF
PRIORITY DOCUMENT

2001年 6月21日

特許庁長官
Commissioner,
Japan Patent Office

及川耕造



【書類名】 特許願

【整理番号】 JP9000420

【提出日】 平成13年 2月28日

【あて先】 特許庁長官 殿

【国際特許分類】 G06F 5/06

【発明者】

【住所又は居所】 神奈川県大和市下鶴間1623番地14 日本アイ・ピー・エム株式会社 東京基礎研究所内

【氏名】 川人 基弘

【発明者】

【住所又は居所】 神奈川県大和市下鶴間1623番地14 日本アイ・ピー・エム株式会社 東京基礎研究所内

【氏名】 小松 秀昭

【発明者】

【住所又は居所】 神奈川県大和市下鶴間1623番地14 日本アイ・ピー・エム株式会社 東京基礎研究所内

【氏名】 ジョン・ファレイ

【特許出願人】

【識別番号】 390009531

【氏名又は名称】 インターナショナル・ビジネス・マシーンズ・コーポレーション

【代理人】

【識別番号】 100086243

【弁理士】

【氏名又は名称】 坂口 博

【代理人】

【識別番号】 100091568

【弁理士】

【氏名又は名称】 市位 嘉宏

【代理人】

【識別番号】 100106699

【弁理士】

【氏名又は名称】 渡部 弘道

【復代理人】

【識別番号】 100104880

【弁理士】

【氏名又は名称】 古部 次郎

【選任した復代理人】

【識別番号】 100100077

【弁理士】

【氏名又は名称】 大場 充

【手数料の表示】

【予納台帳番号】 081504

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【包括委任状番号】 9706050

【包括委任状番号】 9704733

【包括委任状番号】 0004480

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 プログラムの最適化方法及びこれを用いたコンパイラ

【特許請求の範囲】

【請求項 1】 プログラミング言語で記述されたプログラムのソースコードを機械語に変換し、最適化を行うプログラムの最適化方法において、

前記プログラムにおける所定の命令のパラメータを特定の状態に固定することにより当該プログラムの実行速度を向上させることができるかを解析するステップと、

前記解析の結果に基づいて、前記プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成するステップとを含むことを特徴とするプログラムの最適化方法。

【請求項 2】 前記パスを生成するステップは、

前記プログラムを実行することにより、前記解析の結果における前記命令のパラメータが取り得る状態ごとの出現頻度の統計を取るステップと、

得られた統計情報に基づいて、前記パスを生成するステップとを含む請求項 1 に記載のプログラムの最適化方法。

【請求項 3】 プログラミング言語で記述されたプログラムのソースコードを機械語に変換し、最適化を行うプログラムの最適化方法において、

前記プログラムを実行することにより、前記プログラムにおける所定の命令のパラメータが取り得る状態ごとの出現頻度の統計を取るステップと、

得られた統計情報に基づいて、コンパイル結果として所定の命令のパラメータを特定の状態に固定したパスを含む機械語プログラムを生成するステップとを含むことを特徴とするプログラムの最適化方法。

【請求項 4】 前記統計情報に基づいてパラメータを特定の状態に固定できない場合に、コンパイル結果としてパラメータの状態を固定したパスを含まない機械語プログラムを生成するステップをさらに含む請求項 3 に記載のプログラムの最適化方法。

【請求項 5】 オブジェクト指向プログラミング言語で記述されたプログラムのソースコードを機械語に変換し、最適化を行うプログラムの最適化方法にお

いて、

前記プログラムにおける命令のうちで、メソッドの呼び出し先を限定でき、かつ当該メソッドの呼び出し先を限定することにより処理が速くなる命令を検出するステップと、

検出された前記命令におけるメソッドの呼び出し先を、当該命令の処理が速くなるように限定したパスを生成するステップと
を含むことを特徴とするプログラムの最適化方法。

【請求項 6】 プログラミング言語で記述されたプログラムのソースコードを機械語に変換し、最適化を行うプログラムの最適化方法において、

前記プログラムにおける命令のうちで、変数の値を所定の定数値に限定でき、かつ当該定数値に限定することにより処理が速くなる命令を検出するステップと

検出された前記命令における変数を、前記定数値に固定したパスを生成するステップと
を含むことを特徴とするプログラムの最適化方法。

【請求項 7】 プログラミング言語で記述されたプログラムのソースコードを機械語に変換し、最適化を行うコンパイラにおいて、

前記プログラムにおける所定の命令のパラメータを特定の状態に固定することにより当該プログラムの実行速度をどれだけ向上させることができるかを解析する影響度解析部と、

前記影響度解析部による解析結果に基づいて、前記プログラム中に、所定の命令のパラメータを特定の状態に固定した特殊化されたパスを生成する特殊化処理部と

を備えることを特徴とするコンパイラ。

【請求項 8】 前記プログラムを実行した場合に、前記影響度解析部にて解析された前記命令のパラメータが取り得る状態ごとの出現頻度の統計を取り、得られた統計情報に基づいて当該命令のパラメータをどの状態に固定するかを決定する特殊化データ選択部をさらに備え、

前記特殊化処理部は、前記命令のパラメータを、前記特殊化データ選択部によ

り決定された状態に固定した特殊化されたパスを生成する請求項 7 に記載のコンパイラ。

【請求項 9】 前記特殊化処理部は、前記プログラム中に、プログラムの実行状態に応じて、前記特殊化されたパスと特殊化されていないパスとを選択的に実行するための分岐処理を生成し、

前記特殊化データ選択部は、前記分岐処理の挿入に基づく遅延を考慮して前記命令のパラメータをどの状態に固定するかを決定する請求項 8 に記載のコンパイラ。

【請求項 1 0】 プログラムのソースコードを入力する入力装置と、
入力された前記プログラムをコンパイルして機械語コードに変換するコンパイラと、

機械語コードに変換された前記プログラムを実行する処理装置とを備えたコンピュータ装置において、

前記コンパイラは、

前記プログラムにおける所定の命令のパラメータを特定の状態に固定することにより当該プログラムの実行速度を向上させることができるかを解析する手段と

前記解析の結果に基づいて、前記プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成してコンパイルを行う手段とを備え、

前記パラメータの状態を固定したパスを含むコンパイル結果である機械語コードに変換されたプログラムを出力することを特徴とするコンピュータ装置。

【請求項 1 1】 プログラムのソースコードを入力する入力装置と、
入力された前記プログラムをコンパイルして機械語コードに変換するコンパイラと、

機械語コードに変換された前記プログラムを実行する処理装置とを備えたコンピュータ装置において、

前記コンパイラは、

前記プログラムを実行した場合に、前記プログラムにおける所定の命令のパラメータが取り得る状態ごとの出現頻度の統計を取り、当該統計の結果に基づいて

当該命令のパラメータをどの状態に固定するかを決定する手段と、

前記命令のパラメータを、決定された前記状態に固定した特殊化されたパスを生成してコンパイルを行う手段とを備え、

前記特殊化されたパスを含むコンパイル結果である機械語コードに変換されたプログラムを出力することを特徴とするコンピュータ装置。

【請求項 1 2】 前記コンパイラは、

前記特殊化されたパスを生成することなくコンパイルを行う手段をさらに備え

前記コンパイラにおける前記命令のパラメータの状態を決定する前記手段は、固定すべきパラメータの状態を決定できない場合に、前記特殊化されたパスを生成することなくコンパイルを行う前記手段により生成された、前記特殊化されたパスを含まないコンパイル結果である機械語コードに変換されたプログラムを出力する請求項 1 1 に記載のコンピュータ装置。

【請求項 1 3】 コンピュータを制御して実行プログラムの生成を支援する支援プログラムであって、

前記実行プログラムにおける所定の命令のパラメータを特定の状態に固定することにより当該実行プログラムの実行速度を向上させることができるかを解析する機能と、

前記解析の結果に基づいて、前記実行プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成する機能とを前記コンピュータに実現させることを特徴とする支援プログラム。

【請求項 1 4】 コンピュータを制御して実行プログラムの生成を支援する支援プログラムであって、

前記実行プログラムを実行することにより、前記実行プログラムにおける所定の命令のパラメータが取り得る状態ごとの出現頻度の統計を取る機能と、

得られた統計情報に基づいて、前記実行プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成する機能とを前記コンピュータに実現させることを特徴とする支援プログラム。

【請求項 1 5】 コンピュータを制御して実行プログラムの生成を支援する

支援プログラムを当該コンピュータの入力手段が読取可能に記憶した記憶媒体において、

前記支援プログラムは、

実行プログラムにおける所定の命令のパラメータを特定の状態に固定することにより当該実行プログラムの実行速度を向上させることができるかを解析する機能と、

前記解析の結果に基づいて、前記実行プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成する機能と
を前記コンピュータに実現させることを特徴とする記憶媒体。

【請求項 16】 コンピュータを制御して実行プログラムの生成を支援する支援プログラムを当該コンピュータの入力手段が読取可能に記憶した記憶媒体において、

前記支援プログラムは、

前記実行プログラムを実行することにより、前記実行プログラムにおける所定の命令のパラメータが取り得る状態ごとの出現頻度の統計を取る機能と、

得られた統計情報に基づいて、前記実行プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成する機能と
を前記コンピュータに実現させることを特徴とする記憶媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、コンピュータプログラムの最適化方法に関し、特に実行時のデータの性質に基づくプログラムの特殊化を行う最適化方法に関する。

【0002】

【従来の技術】

通常、プログラミング言語で記述されたプログラムのソースコードをコンパイルする際、コンピュータにおける実行速度の向上を図るために、当該プログラムの最適化を行っている。

最適化の手法には種々の方法があるが、オブジェクト指向プログラミング言語

で記述されたプログラムにおいては、プログラムの実行時のデータの性質に基づいてプログラムを特殊化 (specialization) することによる最適化手法がある。

【 0 0 0 3 】

従来のプログラムの特殊化の手法としては、1. スタティック・コンパイラ上でコンパイル時のみ行われる特殊化と、2. スタティック・コンパイラ上でバインディング (Binding) 時及び実行時に行われる特殊化とがある。

前者の例としては、

Flow-Sensitive Interprocedural Constant Propagation :

PLDI (Programming Language Design and Implementation) '95

があり、後者の例としては、

Efficient Incremental Run-Time Specialization for Free : PLDI '99

がある。

【 0 0 0 4 】

スタティック・コンパイラ上でコンパイル時のみ行われる特殊化では、まず、プログラム内のメソッドコールの引数について解析し、「必ず同じ定数値がくると判断できる引数」を検出する。そして、検出された「必ず同じ定数値がくると判断できる引数」を用いて特殊化を行う。

【 0 0 0 5 】

一方、スタティック・コンパイラ上でバインディング (Binding) 時及び実行時に行われる特殊化では、ループネストに特化してプログラムを特殊化する。すなわち、各ループネストのボディ (body) 内で不変とわかる変数について特殊化を行う。この方法は、ネストごとにプログラムを特殊化するという特徴はあるが、基本的には上記のコンパイル時のみ行う手法と同様に、メソッドコールを行う場所で特殊化したメソッドコールを行う。

【 0 0 0 6 】

【発明が解決しようとする課題】

しかしながら、上述した従来のプログラムの特殊化による最適化手法は、いずれもメソッドコールを行う場所で呼び出しメソッドを特殊化するため、引数のパターンに対応する多くの特殊化したメソッドが必要となり、この特殊化したメソ

ッドに対応する実行コードが生成される。このため、コンパイル時間が増加し、かつメモリの消費量が増大していた。

【0007】

また、上記従来のプログラムの特殊化による最適化手法は、上述したコンパイル時間の増加、メモリ消費量の増大といった観点から、J a v aにおけるJ I Tコンパイラ（Just In Timeコンパイラ）のようなダイナミック・コンパイラ（プログラムの実行時に動作するコンパイラ）には適用することが困難であった。

【0008】

さらに、J a v aで用意されているダイナミック・コール（Dynamic Call）のような機能を用いたプログラムでは、メソッドコールを行う場所で呼び出しメソッドが特定できない場合が多い。そのため、メソッドコールを行う場所で呼び出しメソッドを特殊化する上記従来のプログラムの特殊化は実用的ではなかった。

【0009】

そこで、本発明は、特殊化による最適化を含むコンパイルにおいて、コンパイルに要する時間を短縮し、コンパイル時のメモリ消費量を抑えることを目的とする。

また、本発明は、メソッドコールを行う場所で呼び出しメソッドが特定できない場合であっても特殊化を実行可能とすることを他の目的とする。

【0010】

【課題を解決するための手段】

上記の目的を達成するため、本発明は、プログラミング言語で記述されたプログラムのソースコードを機械語に変換し、最適化を行うプログラムの最適化方法において、最適化の対象であるプログラムにおける所定の命令のパラメータを特定の状態に固定することによりこのプログラムの実行速度を向上させることができるかを解析するステップと、この解析結果に基づいて、このプログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成して特殊化するステップとを含むことを特徴とする。

本発明では、この所定の命令のパラメータを特定の状態に固定したパス（特殊化されたパス）と、この所定の命令のパラメータを特定の状態に固定しないパス

(特殊化されていないパス)とを、分岐処理により選択的に実行させるように、プログラムの最適化を行う。これにより、特殊化されたパスを実行した際にプログラムの実行速度を向上させることができ、かつこの特殊化されたパスを実行する割合が大きいならば、プログラム全体の実行効率を向上させることができる。

【0011】

ここで、このパスを生成するステップは、このプログラムを実行することにより、この解析結果における前記命令のパラメータが取り得る状態ごとの出現頻度の統計を取るステップと、得られた統計情報に基づいて、この所定の命令のパラメータを特定の状態に固定したパスを生成するステップとを含む。

このように、統計処理を行うことにより、特殊化されたパスを実行する確率がどの程度あるかを的確に調べることができる。

【0012】

また、本発明は、プログラムの最適化方法において、最適化の対象であるプログラムを実行することにより、このプログラムにおける所定の命令のパラメータが取り得る状態ごとの出現頻度の統計を取るステップと、得られた統計情報に基づいて、コンパイル結果として所定の命令のパラメータを特定の状態に固定したパスを含む機械語プログラムを生成するステップとを含むことを特徴とする。

【0013】

さらに、この最適化方法は、この統計情報に基づいてパラメータを特定の状態に固定できない場合に、コンパイル結果としてパラメータの状態を固定したパスを含まない機械語プログラムを生成するステップを含むことができる。

【0014】

また、本発明は、上記のような特殊化によるプログラムの最適化方法において、最適化の対象であるプログラムにおける命令のうちで、メソッドの呼び出し先を限定でき、かつこのメソッドの呼び出し先を限定することにより処理が速くなる命令を検出するステップと、検出された命令におけるメソッドの呼び出し先を、この命令の処理が速くなるように限定したパスを生成するステップとを含むことを特徴とする。

【0015】

あるいは、上記のような特殊化によるプログラムの最適化方法において、最適化の対象であるプログラムにおける命令のうちで、変数の値を所定の定数値に限定でき、かつこの定数値に限定することにより処理が速くなる命令を検出するステップと、検出された命令における変数を、この定数値に固定したパスを生成するステップとを含むことを特徴とする。

【 0 0 1 6 】

また、本発明は、プログラミング言語で記述されたプログラムのソースコードを機械語に変換し、最適化を行うコンパイラにおいて、コンパイルの対象であるプログラムにおける所定の命令のパラメータを特定の状態に固定することによりこのプログラムの実行速度をどれだけ向上させることができるかを解析する影響度解析部と、この影響度解析部による解析結果に基づいて、このプログラム中に、所定の命令のパラメータを特定の状態に固定した特殊化されたパスを生成する特殊化処理部とを備えることを特徴とする。

【 0 0 1 7 】

さらに、このコンパイラは、このプログラムを実行した場合に、この影響度解析部にて解析された命令のパラメータが取り得る状態ごとの出現頻度の統計を取り、得られた統計情報に基づいてこの命令のパラメータをどの状態に固定するかを決定する特殊化データ選択部をさらに備える構成とすることができる。そして、特殊化処理部は、この命令のパラメータを、特殊化データ選択部により決定された状態に固定した特殊化されたパスを生成する。

【 0 0 1 8 】

ここで、この特殊化処理部は、このプログラム中に、プログラムの実行状態に応じて、特殊化されたパスと特殊化されていないパスとを選択的に実行するための分岐処理を生成し、特殊化データ選択部は、この分岐処理の挿入に基づく遅延を考慮してこの命令のパラメータをどの状態に固定するかを決定する構成とすることができる。

【 0 0 1 9 】

さらにまた、本発明は、次のように構成されたことを特徴とするコンピュータ装置として提供することができる。すなわち、プログラムのソースコードを入力

する入力装置と、入力されたこのプログラムをコンパイルして機械語コードに変換するコンパイラと、機械語コードに変換されたこのプログラムを実行する処理装置とを備える。また、このコンパイラは、このプログラムにおける所定の命令のパラメータを特定の状態に固定することによりこのプログラムの実行速度を向上させることができるかを解析する手段と、この解析結果に基づいて、このプログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成してコンパイルを行う手段とを備える。そして、このコンパイラは、このパラメータの状態を固定したパスを含むコンパイル結果である機械語コードに変換されたプログラムを出力する。

【 0 0 2 0 】

また、本発明は、次のように構成されたことを特徴とするコンピュータ装置として提供することができる。すなわち、プログラムのソースコードを入力する入力装置と、入力されたこのプログラムをコンパイルして機械語コードに変換するコンパイラと、機械語コードに変換されたこのプログラムを実行する処理装置とを備える。また、このコンパイラは、このプログラムを実行した場合に、このプログラムにおける所定の命令のパラメータが取り得る状態ごとの出現頻度の統計を取り、この統計の結果に基づいてこの命令のパラメータをどの状態に固定するかを決定する手段と、この命令のパラメータを、決定された状態に固定した特殊化されたパスを生成してコンパイルを行う手段とを備える。そして、このコンパイラは、この特殊化されたパスを含むコンパイル結果である機械語コードに変換されたプログラムを出力する。

【 0 0 2 1 】

さらに、このコンパイラは、この特殊化されたパスを生成することなくコンパイルを行う手段をさらに備える構成とすることができる。そして、このコンパイラにおける命令のパラメータの状態を決定する手段は、固定すべきパラメータの状態を決定できない場合に、この特殊化されたパスを生成することなくコンパイルを行う手段により生成された、この特殊化されたパスを含まないコンパイル結果である機械語コードに変換されたプログラムを出力する。

【 0 0 2 2 】

また、本発明は、次のように構成されたことを特徴とするコンピュータプログラムとして提供することができる。すなわち、コンピュータを制御して実行プログラムの生成を支援する支援プログラムであって、この実行プログラムにおける所定の命令のパラメータを特定の状態に固定することによりこの実行プログラムの実行速度を向上させることができるかを解析する機能と、この解析結果に基づいて、この実行プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成する機能とをコンピュータに実現させることを特徴とする。

【 0 0 2 3 】

あるいは、コンピュータを制御して実行プログラムの生成を支援する支援プログラムであって、この実行プログラムを実行することにより、この実行プログラムにおける所定の命令のパラメータが取り得る状態ごとの出現頻度の統計を取る機能と、得られた統計情報に基づいて、この実行プログラム中に、所定の命令のパラメータを特定の状態に固定したパスを生成する機能とをコンピュータに実現させることを特徴とする。

【 0 0 2 4 】

さらに本発明は、かかるコンピュータプログラムをコンピュータの入力手段が読取可能に記憶した記憶媒体として提供したり、かかるコンピュータプログラムをネットワークを介して伝送させるプログラム伝送装置として提供したりすることもできる。

【 0 0 2 5 】

【発明の実施の形態】

以下、添付図面に示す実施の形態に基づいて、この発明を詳細に説明する。

図 1 は、本実施の形態におけるコンパイラの全体構成を説明する図である。

図 1 を参照すると、本実施の形態のコンパイラ 1 0 は、入力プログラムに対して通常のコンパイル処理を行う汎用コンパイル処理部 1 1 と、汎用コンパイル処理部 1 1 によるコンパイル結果に対して影響度解析 (Impact Analysis) を行う影響度解析部 1 2 と、影響度解析部 1 2 による解析結果を用いて特殊化を行うデータを選択する特殊化データ選択部 1 3 と、特殊化データ選択部 1 3 により選択されたデータを特殊化してオブジェクトプログラムを生成する特殊化コンパイル

処理部 1 4 とを備える。

図 1 に示したコンパイラ 1 0 の各構成要素は、コンピュータプログラムにより制御された CPU にて実現される仮想的なソフトウェアブロックである。CPU を制御する当該コンピュータプログラムは、CD-ROM やフロッピーディスクなどの記憶媒体に格納して配布したり、ネットワークを介して伝送したりすることにより提供される。

【 0 0 2 6 】

上記の構成要素において、汎用コンパイル処理部 1 1 は、入力プログラムであるソースプログラムを入力し、通常のコンパイル処理を行って機械語コードによるオブジェクトプログラムを生成する。コンパイラ 1 0 を J a v a における J I T コンパイラにおいて実現する場合は、ソースプログラムはバイトコードで記述されたプログラムとなる。

【 0 0 2 7 】

影響度解析部 1 2 は、汎用コンパイル処理部 1 1 によるコンパイル結果であるオブジェクトプログラムを対象として、影響度解析処理を実行し、プログラム内のデータにおける影響度情報を作成する。影響度解析処理とは、「どのプログラム内のデータを、どのように特殊化すると、より強力な最適化を行えるか」を調べる処理である。影響度情報には、この解析結果が記述される。影響度解析処理の具体的な内容については後述する。

【 0 0 2 8 】

特殊化データ選択部 1 3 は、影響度解析部 1 2 による影響度解析処理の結果、特殊化による影響のあるデータが存在する場合に、特殊化を行うデータを選択する。具体的には、汎用コンパイル処理部 1 1 で生成されたオブジェクトプログラムを複数回実行し、影響度解析部 1 2 で作成された影響度情報についてデータの出現頻度の統計を取る。すなわち、影響度解析処理の結果において特殊化による影響があると判断されたデータが、どのように実行される確率が、どの程度有るかを調べる。そして、得られた統計情報において特殊化を行うことにより一定以上の効果が得られると評価できるデータを特殊化の対象に決定する。特殊化の対象となるデータを選択する処理の具体的な内容については後述する。

ここで、統計を取るためには、当該プログラムが実際に実行されることが必要である。すなわち、当該プログラムを実行するシステムは、初期的には、汎用コンパイル処理部 11 による通常のコンパイルを経たオブジェクトプログラムを実行する。そして、十分な回数実行して統計が得られた後に（統計を取るための実行回数は任意に設定できる）、特殊化の対象となるデータの選択が行われることとなる。

ただし、この統計を取るための実行は、当該統計を取ることを目的とした試験的な実行であっても良い。この場合、当該プログラムが実際のシステムで使用される場合には最初から適切な特殊化が行われた状態でコンパイルされることとなる。

【0029】

なお、影響度解析部 12 による影響度解析処理の結果、特殊化による影響のあるデータが存在しない場合、特殊化データ選択部 13 は、上述した統計処理を行うことなく、汎用コンパイル処理部 11 により生成されたオブジェクトプログラムを出力する。

また、特殊化データ選択部 13 による統計処理の結果において、特殊化すべきデータが存在しない場合、またはプログラムの実行回数が統計処理のために設定された回数に満たない場合は、特殊化コンパイル処理部 14 に移行して特殊化を行うことなく、汎用コンパイル処理部 11 により生成されたオブジェクトプログラムを出力する。

【0030】

特殊化コンパイル処理部 14 は、特殊化データ選択部 13 により特殊化の対象として選択されたデータを特殊化し、特殊化されたオブジェクトプログラムを生成して出力する。特殊化は、プログラム中の所定の命令におけるパラメータを特定の状態（定数化や呼び出し先の特定など）に固定することにより、パラメータの状態を判断する処理のようなパラメータが複数の状態を取ることに伴う処理を省略するものであり、対象であるデータの種類や性質に応じて個別的に実行される。例えば、命令中の定数を最も頻出する値に固定したり、メソッドの呼び出し先を最も多い呼び出し先に特定したりするといった処理が行われる。

【0031】

図2は、本実施の形態によるコンパイラ10の全体的な処理の流れを説明するフローチャートである。

図2を参照すると、まず、ソースプログラムが汎用コンパイル処理部11に入力され、当該ソースプログラムに対して通常のコンパイル処理が実行される（ステップ201）。

次に、汎用コンパイル処理部11によるコンパイル結果であるオブジェクトプログラムに対して、影響度解析部12により影響度解析が行われる（ステップ202）。そして、特殊化による影響のあるデータが存在しない場合は、汎用コンパイル処理部11により生成されたオブジェクトプログラムが出力されて、処理が終了する（ステップ203）。この場合、当該プログラムを実行するシステムは、汎用コンパイル処理部11のみによってコンパイルされたオブジェクトプログラムを実行することとなる。

【0032】

影響度解析において特殊化による影響のあるデータが存在すると判断された場合は、次に、特殊化データ選択部13により、影響度解析処理の結果において特殊化による影響があると判断されたデータに対する統計処理が行われ、特殊化の対象となるデータの選択が行われる（ステップ203、204）。そして、特殊化すべきデータが存在しない場合は、汎用コンパイル処理部11により生成されたオブジェクトプログラムが出力されて、処理が終了する（ステップ205）。

ここで、統計処理の結果、特殊化すべきデータが存在しない場合は、当該プログラムを実行するシステムは、汎用コンパイル処理部11のみによってコンパイルされたオブジェクトプログラムを実行することとなる。

一方、プログラムの実行回数が統計処理のために設定された回数に満たないことにより特殊化すべきデータが選択されない場合は、当該プログラムがさらに実行されて統計処理を行うに十分な回数となるのを待つ。

【0033】

プログラムの実行回数が統計処理のために設定された回数に達し、統計が取られた結果、特殊化すべきデータが存在する場合は、次に、特殊化コンパイル処理

部 1 4 により、当該データの特特殊化が行われる（ステップ 2 0 5、2 0 6）。

これにより、当該プログラムを実行するシステムは、特殊化コンパイル処理部 1 4 により特殊化されたオブジェクトプログラムを実行することとなる。

【0 0 3 4】

次に、上述した影響度解析部 1 2 による影響度解析処理の内容及び特殊化データ選択部 1 3 による特殊化の対象となるデータを選択する処理の内容について説明する。

ここでは、J a v a 言語に対して本実施の形態を適用した場合を例として、パラメータを特殊化する例と、グローバルデータの両方を特殊化する例の 2 つを挙げる。なお、以下の説明において、ALLTYPEとは実装を行っている全ての特殊化するタイプを意味する。図 3 にこの特殊化するタイプの例を示す。

【0 0 3 5】

図 3 に示す例では、特殊化のタイプとして、「定数」、「N u l l ではない」、「クラスの特特定」、「配列以外のクラスである」といったタイプが明示されている。ここで、「定数」は、プログラムにおける所定の命令中の定数を一定の値に固定する特殊化を行う。「N u l l ではない」は、変数の値が N u l l となる可能性があり、かつ N u l l であれば処理が速くなる場合に、当該変数の値を N u l l に固定する特殊化を行う。「クラスの特特定」は、メソッドの呼び出し先を限定でき、かつメソッドの呼び出し先を限定することにより処理が速くなる場合に、当該メソッドの呼び出し先を特定する特殊化を行う。「配列以外のクラスである」は、プログラムが J a v a のようにメソッドの呼び出し先として配列クラスを取ることができるプログラム言語で記述されている場合に、当該メソッドの呼び出し先を配列クラスに特定する特殊化を行う。なお、特殊化のタイプは、プログラムの種類や実装するシステム、使用環境などに応じて種々のタイプを設定することができる。

【0 0 3 6】

また、図 3 の命令欄には、これらの特殊化を行う可能性のある命令の種類が記載されている。例えば、プログラム中に四則演算やオブジェクト以外の変数に対する比較命令が含まれていたならば、これらの命令は「定数」タイプの特殊化を

行うことにより処理効率が向上する可能性がある。同様に、プログラム中に INSTANCEOF、CHECKCAST、仮想的なメソッド呼び出し命令などが含まれていたならば、これらの命令は「クラスの特化」タイプの特殊化を行うことにより処理効率が向上する可能性がある。したがって、これらの命令は影響度解析部 1 2 による影響度解析処理の対象となる。

【 0 0 3 7 】

(1) パラメータの特殊化

まず、パラメータを特殊化する場合における本実施の形態の動作アルゴリズムを説明する。

最初に、汎用コンパイル処理部 1 1 が、入力されたソースプログラムに対して通常の（特殊化を行わない）コンパイル処理を行う（図 2、ステップ 2 0 1 参照）。このとき、UD-chain (use-definition chain: 使用-定義連鎖) 及び DU-chain (definition-use chain: 定義-使用連鎖) も作成しておく。

【 0 0 3 8 】

次に、影響度解析部 1 2 が、汎用コンパイル処理部 1 1 によるコンパイル結果（特殊化されていないオブジェクトプログラム）に対して影響度解析処理を行う（図 2、ステップ 2 0 2 参照）。

影響度解析処理としては、まず、プログラム中のパラメータ（変数）に対して、予め影響度解析の対象として設定された特殊化の各タイプ（図 3 参照）に対して、特殊化を行った場合における効果を見積もる。

【 0 0 3 9 】

ここで、特殊化を行った場合の効果について説明する。

特殊化を行うとは、ここでは、複数の値を取る可能性のあるパラメータを含む命令に関して、当該パラメータが特定の値を取るものとしてプログラムを書き換えることを意味する。これに伴い、当該パラメータが当該特定の値以外の値を取る場合に対応するため、当該命令の前にガードコードを挿入し、特殊化されていないパスへの分岐処理を行う必要が生じる。したがって、パラメータを特殊化しても特殊化したパスの実行速度が特殊化前と変わらない場合、ガードコードを挿

入した分だけプログラム全体の実行速度は遅くなる。

一方、特殊化したパスの実行速度が特殊化前よりも速くなるならば、ガードコードを挿入したことによる遅延分を越えてプログラム全体の実行速度が向上する可能性がある。

【0040】

そこで、特殊化を行った場合の効果を見積もり、これとガードコードの挿入による遅延を基準とするしきい値Aとを比較することにより、プログラム全体の実行速度に対する影響の度合いを調べる。この影響の度合いは、例えば、ガードコードの挿入による遅延に対する特殊化の効果の比で表すことができる（以下、この値を影響度値と呼ぶ）。この場合、影響度値が1であるとは、特殊化されたパスの実行速度の向上分がガードコードの挿入による遅延分と等しい（すなわち、プログラム全体の実行速度は変わらない）ことを意味する。

そして、特殊化されたパスの実行速度の向上がプログラム全体の実行速度に対して一定以上の影響を与える場合に、当該特殊化におけるパラメータ及び特定の値を、特殊化による影響のあるデータ（以下、特殊化対象データ）として影響度情報に記述する。また、影響度情報には、かかる特殊化におけるプログラム全体の実行速度に対する影響度値を記述する。

【0041】

なお、しきい値Aを適当に設定することにより、プログラム全体の実行速度に対してどの程度の影響を与える場合に特殊化対象データを影響度情報に記述するかを調節することができる。しきい値Aを小さく設定すれば、わずかでもプログラム全体の実行速度を向上させる特殊化対象データが、影響度情報に記述されることとなる。一方、しきい値Aを大きく設定すれば、相当程度プログラム全体の実行速度を向上させる特殊化対象データでなければ、影響度情報に記述されない。しかしこの場合、影響度情報に記述される特殊化対象データの数が減るので、特殊化データ選択部13による統計処理のコストを削減し、特殊化データ選択部13を実現するCPUの負担を軽減することができる。

【0042】

図4は、上述したパラメータに対する特殊化における影響度解析処理のアルゴ

リズムを説明する図である。

図4のアルゴリズムにおける6行目の

“Estimate(impact, p, p, ϕ);”

において特殊化の効果を見積もる。かかる処理の具体的な内容（アルゴリズム）は図5に示す。

【0043】

また、図4のアルゴリズムにおける8、9行目の

“if (Impact[p][type] > しきい値A)”

“Effective \cup = { p, type } ;”

において特殊化の効果としきい値Aとを比較し、影響度情報に記述する情報を決定している。8行目のコメントにあるように、しきい値Aの値を1とすれば、ガードコードの挿入による遅延分だけ特殊化されたパスの実行速度が向上する。したがって、プログラム全体の実行速度を向上させるには、しきい値Aの値を1よりも大きくする必要がある。

【0044】

次に、特殊化データ選択部13が、影響度解析部12による影響度解析処理の結果に基づいて、特殊化を実行するデータを選択する（図2、ステップ203、204参照）。

図6は、特殊化データ選択部13による統計処理のアルゴリズムを説明する図である。

図6において、3行目にあるように、影響度情報に記述されたパラメータ（p）に対して、対応する特殊化の方法（type）に固有の条件で統計を取る。また、1行目にあるように、影響度情報（Effective）にデータが記述されていない場合（ ϕ 空集合）は、統計処理を行わない。

【0045】

次に、特殊化データ選択部 13 は、統計処理により得られた統計情報に基づいて、特殊化を実行するデータ（以下、特殊化実行データ）を選択する。

上述したように、特殊化とは、複数の値を取る可能性のあるパラメータを含む命令に関して、当該パラメータが特定の値を取るものとしてプログラムを書き換えることである。したがって、パラメータが取り得る値は複数あり、その個数だけ特殊化のバリエーションが存在することとなる。そこで、特殊化のバリエーションごとに出現頻度の統計を取り、パラメータが取り得る値のうちで最も頻出する値を選び、これを特殊化実行データとする。

ただし、最も頻出する値であっても、全体に対して十分大きな確率で出現しなければ、プログラム全体の処理効率が向上しないと考えられる場合がある。すなわち、パラメータの取り得る値が多数ある場合、各値が同程度の確率で出現するならば、最も頻出する値であっても出現率が大きくなりえない場合があり得る。この場合、特殊化を行ったとしても、特殊化されていないパスが実行される確率が高いため、ガードコードを挿入したことによる遅延がプログラム全体の実行速度に大きく影響することとなる。

したがって、ここでは、パラメータが取り得る値のうちで最も頻出する値について、このパラメータがこの最頻出値を取る確率を統計情報から取得し、この最頻出値が出現する確率と、この最頻出値で特殊化した場合における影響度値とを乗じた値を算出する。そして、得られた乗算値とガードコードの挿入による遅延を基準とするしきい値 B とを比較することにより、プログラム全体の処理効率が向上するかどうかを調べる。これにより、一定以上の効果が得られる場合に、当該最頻出値を上述した特殊化実行データとして最終的に選択する。

【0046】

なお、しきい値 B を適当に設定することにより、特殊化によるプログラムの最適化が当該プログラム全体の実行効率に対してどの程度の影響を与えるかを制御することができる。しきい値 B を小さく設定すれば、上述した考え方に基づいて、特殊化を行わない場合よりもわずかでも実行効率が上がると考えられる場合に特殊化を行うこととなる。一方、しきい値 B を大きく設定すれば、相当程度プログラムの実行効率を向上させると考えられる場合でなければ特殊化を行わない。

しかしこの場合、特殊化を行う箇所を特殊化の効果の大きい場所に制限することとなるため、コンパイルの際のCPUの負担を軽減し、かつ効果の大きい最適化を実現することができる。そのため、しきい値Bを適切に設定することにより、コンパイルに要する時間をあまり増大させることなく特殊化を実行することができる。JavaにおけるJITコンパイラのような、プログラムを実行する際にコンパイルを行うダイナミック・コンパイラに対しても適用することが可能となる。

【0047】

特殊化データ選択部13により上記のようにして決定される特殊化実行データと、影響度解析部12により決定される特殊化対象データとを比較すると、影響度解析部12により決定される特殊化対象データは、特殊化することによってプログラム全体の実行速度の向上が見込まれる全てのパラメータに関するデータであり、特殊化データ選択部13により決定される特殊化実行データは、特殊化対象データのうちでプログラム全体の実行効率を最も向上させるパラメータに関するデータである。

【0048】

図7は、上述したパラメータに対する特殊化における特殊化実行データを選択する処理のアルゴリズムを説明する図である。

図7のアルゴリズムにおける3行目の

“odds = { p, type } に関する統計の中で、もっとも優秀だった確率;”

において、パラメータが取り得る値の最頻出値の出現率を求める。また、4行目の

“val = oddsに対応する { p, type } の値;”

において、最頻出値自体を求める。そして、5、6行目の

“if (Impact[p] [type] * odds > しきい値B)”

“Specialize U= { p, type, val } ;”

において、最頻出値の出現率とこの最頻出値に対応する影響度値とを乗算し、その乗算値としきい値Bとを比較し、特殊化を実行するデータを決定している。

なお、9行目にあるように、所定のパラメータに関して特殊化を実行するデータ (Specialize) が決定されなかった場合 (空集合) は、当該パラメータに対する特殊化コンパイル処理部14による特殊化は行わない (図2、ステップ205参照)。

【0049】

最後に、特殊化コンパイル処理部14が、特殊化データ選択部13により選択された特殊化実行データに対して、実際に特殊化を実行する。特殊化は、上述したように、パラメータの種類や特殊化実行データの値に応じて個別的に実行される。

【0050】

以上のように、本実施の形態では、特殊化によりプログラム全体の実行速度が向上する程度とかかる特殊化されたプログラムが実行される頻度とに基づいて、当該プログラム全体の実行効率を考える。このため、従来の特許化において、所定のパラメータが必ず定数値を取る場合など極めて限定された場合でのみ特殊化が可能であったのに対し、本実施の形態は、特殊化によってプログラム全体の実行効率が一定以上向上すると考えられる場合には特殊化を行うことができる。

また、従来、プログラム中の特殊化できる可能性のある全てのパラメータに対して特殊化が可能かどうかを調べて特殊化を実行していたのに対し、本実施の形態は、特殊化コンパイル処理部14により特殊化を実行する前に、影響度解析部12による影響度解析処理を行い、さらに特殊化データ選択部13により特殊化の対象となる値の実際の出現頻度を統計処理するという二重の処理により、特殊化の対象とするパラメータを絞り込んでいる。これによって、特殊化を行う場合のコンパイル時間及びメモリ消費量の増加を少なく抑えることができる。

さらに、本実施の形態は、メソッドコールを行う場所で呼び出しメソッドを特

特殊化するのではなく、実際に呼ばれたメソッドについて解析して特殊化を行う。このため、メソッドコールを行う場所で呼び出しメソッドを特定できない場合であっても、特殊化を行うことができる。

【 0 0 5 1 】

次に、上述したパラメータの特殊化の適用例を説明する。

図 8 は、本適用例に用いるサンプルプログラム（以下、第 1 のサンプルプログラムと記す）を示す図である。

コンパイラ 1 0 に図 8 の第 1 のサンプルプログラムが入力されると、まず、汎用コンパイル処理部 1 1 により、当該第 1 のサンプルプログラムに対して通常のコンパイルが行われる。ただし、第 1 のサンプルプログラムは、この通常のコンパイルにおける最適化では、特に変更される箇所はない。なお、これ以降、第 1 のサンプルプログラムはコンパイルの過程で最適化が行われるため、コンパイル結果は機械語で記述されたオブジェクトプログラムとなっている。しかしながら、説明の簡単のため、ここではソースプログラムの状態のままで記述する。以下の他の適用例についても同様である。

【 0 0 5 2 】

次に、影響度解析部 1 2 により、汎用コンパイル処理部 1 1 によるコンパイル結果に対して影響度解析処理が行われ、特殊化による効果が見込まれる特殊化対象データが抽出される。

図 9 は、図 8 に示した第 1 のサンプルプログラムに対する影響度解析処理の結果を示す図である。

図 9 (A) を参照すると、パラメータ（図示の p 欄）srcBegin が何らかの定数値を取る場合（「定数」タイプの特殊化を行った場合）に、影響度値 2. 2 5 が得られる。また、パラメータ srcEnd が何らかの定数値を取る場合に、影響度値 1. 2 5 が得られる。したがって、図 9 (A) に提示された第 1 のサンプルプログラム中の 4 つのパラメータのうち、srcBegin 及び srcEnd の 2 つについて、定数値の出現頻度の統計を取ればよいことがわかる。そして、この 2 つのパラメータに関する情報が特殊化対象データとして影響度情報に記述される。

なお、図 9 (B) には、第 1 のサンプルプログラムのソースプログラム中にお

けるパラメータsrcBegin、srcEndの位置を示してある（太字で記し、下線を付してある）。

【0053】

次に、特殊化データ選択部13により、パラメータsrcBegin、srcEndに関して、取り得る値ごとに出現頻度の統計が取られる。

図10は、得られた統計情報の例を示す図である。なお、図示の例は、第1のサンプルプログラムに対してSPECjvm98の中のJackベンチマークを使った場合における統計結果である。

【0054】

次に、上記の統計情報に基づいて、特殊化データ選択部13により、特殊化実行データが決定される。

図10の統計結果を参照すると、srcBeginについては、定数値として値0を取る確率が100%である。また、srcEndについては、定数値として値1を取る確率が68%、値0を取る確率が15%、値2を取る確率が4%、値3を取る確率が3%、値4を取る確率が3%である。その他の値については出現率が小さいので省略してある。

【0055】

以上の統計情報によれば、srcBeginについては、特定の定数値の出現率が100%であり、その影響度値が2.25であるため、その乗算値は2.25となる。この値はガードコードの挿入による遅延分の1よりも大きい。したがって、srcBeginは特殊化実行データとされる。

一方、srcEndについては、最も頻出する値1の場合で出現率が68%であり、影響度値が1.25であるため、その乗算値は0.85となる。この値はガードコードの挿入による遅延分の1よりも小さい。したがって、プログラム全体の実行効率に対する影響は、srcBeginを特殊化することによる実行速度の向上よりもガードコードを挿入したことによる遅延の方が大きいこととなる。また、1以外の値では、出現率が小さくなるため、プログラム全体の実行効率に対する影響がさらに小さくなることは自明である。

したがって、この適用例では、srcBeginのみが特殊化実行データとなる。

【 0 0 5 6 】

以上の処理の後、特殊化コンパイル処理部 1 4 により、図 8 に示した第 1 のサンプルプログラムにおける srcBegin が値 0 に特殊化されてコンパイルされる。

図 1 1 は、図 8 の第 1 のサンプルプログラムに対して、かかる特殊化が行われた状態のソースプログラムを示す図である。

【 0 0 5 7 】

次に、パラメータの特殊化の他の適用例を説明する。

図 1 2 は、本適用例に用いるサンプルプログラム（以下、第 2 のサンプルプログラムと記す）を示す図である。

コンパイラ 1 0 に図 1 2 の第 2 のサンプルプログラムが入力されると、まず、汎用コンパイル処理部 1 1 により、当該第 2 のサンプルプログラムに対して通常のコンパイルが行われる。

図 1 3 は、図 1 2 の第 2 のサンプルプログラムに対して通常のコンパイルにおける最適化が行われた状態を示す図である。ここでは、checkcast の除去などの若干の最適化が実現されている。

【 0 0 5 8 】

次に、影響度解析部 1 2 により、汎用コンパイル処理部 1 1 によるコンパイル結果に対して影響度解析処理が行われ、特殊化による効果が見込まれる特殊化対象データが抽出される。

図 1 4 は、図 1 2 に示した第 2 のサンプルプログラムに対する影響度解析処理の結果を示す図である。

図 1 4 を参照すると、パラメータ e のクラスが特定されている場合に、影響度値 3.94 が得られる。したがって、このパラメータ e について、クラスの出現頻度の統計を取ればよいことがわかる。そして、このパラメータ e に関する情報が特殊化対象データとして影響度情報に記述される。

なお、図 1 3 のプログラムにおいて太字で記し下線を付した部分が、パラメータ e を含み、特殊化による効果が見込まれる部分である。

【 0 0 5 9 】

次に、特殊化データ選択部 1 3 により、パラメータ e に関して、取り得るクラ

スごとに出現頻度の統計が取られる。そして、得られた統計情報に基づいて特殊化実行データが決定される。具体的な統計情報の提示は省略するが、ここでは、パラメータ *e* が *KeyEvent* というクラスである場合が最も出現率が高く、この場合の出現率と影響度値との乗算値はガードコードの挿入による遅延分の 1 よりも大きいものとする。したがって、パラメータ *e* は特殊化実行データとされる。

【0060】

以上の処理の後、特殊化コンパイル処理部 14 により、図 12 に示した第 2 のサンプルプログラムにおけるパラメータ *e* のクラスが *KeyEvent* に特殊化されてコンパイルされる。

図 15 は、図 12 の第 2 のサンプルプログラムに対して、かかる特殊化が行われた状態のソースプログラムを示す図である。

【0061】

さらにここで、特殊化のタイプとして「クラスの特定 かつ Null でない」という組合せの状態が定義されている場合を考える。この場合、上述した各処理を経て、図 15 における 3 行目の

“if (*e* != null)”

が特殊化されることとなる。図 16 は、かかる特殊化が行われた状態のソースプログラムを示す図である。

【0062】

(2) グローバルデータの特殊化

次に、グローバルデータを特殊化する場合における本実施の形態の同アルゴリズムを説明する。なお、実際のコンパイル処理においては、グローバルデータに対する特殊化と上述したパラメータに対する特殊化とを区別することなく実行するが、ここでは、説明上グローバルデータに対する特殊化のみを説明し、後述する適用例においてパラメータ及びグローバルデータに対する特殊化の例を示す。

【0063】

最初に、汎用コンパイル処理部 11 が、入力されたソースプログラムに対して

通常の（特殊化を行わない）コンパイル処理を行う（図 2、ステップ 2 0 1 参照）。このとき、グローバルデータをメソッドローカルな変数に置き換える Scalar replacement の最適化を行うと共に、UD-chain（使用-定義連鎖）及び DU-chain（定義-使用連鎖）の作成を行っておく。

【 0 0 6 4 】

Scalar replacement の最適化を行うアルゴリズムとして、
Lazy Code Motion : PLDI' 92

に記載されたアルゴリズムがある。このアルゴリズムには、Down-Safety という集合情報を計算する部分がある。これは、メソッド内の式を実行とは逆方向に動かす時に、その式の結果が変わらないと判断できる領域を集合として求めるものである。

したがって、Scalar replacement の最適化に Lazy Code Motion のアルゴリズムを用いる場合、この Down-Safety という集合情報を利用することにより、「メソッドの先頭にグローバルデータのアクセスを移動できる」と判断できるアクセスの集合を求めることができる。そして、この情報をグローバルデータに対する特殊化に用いる。

【 0 0 6 5 】

次に、影響度解析部 1 2 が、汎用コンパイル処理部 1 1 によるコンパイル結果（特殊化されていないオブジェクトプログラム）に対して影響度解析処理を行う（図 2、ステップ 2 0 2 参照）。

影響度解析処理としては、まず、「メソッドの先頭にグローバルデータのアクセスを移動できる集合」が「DownSafety[メソッドの先頭]」に入っていると仮定し、この結果が格納されたローカル変数を定義する。

この後、定義されたローカル変数に対して、予め影響度解析の対象として設定された特殊化の各タイプ（図 3 参照）に対して、特殊化を行った場合における効果を見積もり、特殊化対象データを決定して影響度情報を作成する。これらの動作の内容は、（1）パラメータに対する特殊化の場合と同様である。

【 0 0 6 6 】

図 1 7 は、上述したグローバルデータに対する特殊化における影響度解析処理

のアルゴリズムを説明する図である。

図 1 7 のアルゴリズムにおける 3、4 行目の

“for (each $p \in \text{DownSafety}[\text{メソッドの先頭}]$)”

“lvar = pの結果が格納されたローカル変数;”

においてグローバルデータのローカル変数化を実現している。また、8 行目の

“Estimate(impact, lvar, lvar, ϕ);”

の処理の内容は、図 5 に示したパラメータの特殊化における特殊化の効果を見積もる処理と同様である。

【0 0 6 7】

次に、特殊化データ選択部 1 3 が、影響度解析部 1 2 による影響度解析処理の結果に基づいて、統計処理を行い、特殊化を行うグローバルデータ及びその値（実行データ）を決定する（図 2、ステップ 2 0 3、2 0 4 参照）。これらの処理は、上述したパラメータの特殊化における特殊化データ選択部 1 3 の処理と同様である。

【0 0 6 8】

最後に、特殊化コンパイル処理部 1 4 が、特殊化データ選択部 1 3 により選択された特殊化実行データに対して、実際に特殊化を実行する。特殊化は、上述したように、パラメータの種類や特殊化実行データの値に応じて個別的に実行される。

【0 0 6 9】

次に、パラメータ及びグローバルデータの特殊化の適用例を説明する。

図 1 8 は、本適用例に用いるサンプルプログラム（以下、第 3 のサンプルプログラムと記す）を示す図である。

コンパイラ 1 0 に図 1 8 の第 3 のサンプルプログラムが入力されると、まず、汎用コンパイル処理部 1 1 により、当該第 3 のサンプルプログラムに対して通常

のコンパイルが行われる。

図 1 9 は、図 1 8 の第 3 のサンプルプログラムに対して通常のコンパイルにおける最適化が行われた状態を示す図である。

【 0 0 7 0 】

次に、影響度解析部 1 2 により、汎用コンパイル処理部 1 1 によるコンパイル結果に対して影響度解析処理が行われ、特殊化による効果が見込まれる特殊化対象データが抽出される。ここでは、上述したパラメータに対する影響度解析処理と、グローバルデータに対する影響度解析処理とが区別なく実行される。

図 2 0 は、図 1 9 に示したサンプルプログラム 3 のコンパイル結果に対する影響度解析処理の結果を示す図である。

図 2 0 を参照すると、パラメータ `fromIndex` が何らかの定数値を取る場合に、影響度値 1. 7 5 が得られる。また、パラメータ `this. offset` が何らかの定数値を取る場合に、影響度値 5. 7 5 が得られる。また、パラメータ `this. count` が何らかの定数値を取る場合に、影響度値 5. 7 5 が得られる。したがって、図 2 0 に提示された 5 つのパラメータのうち、`fromIndex`、`this. offset` 及び `this. count` の 3 つについて、定数値の出現頻度の統計を取ればよいことがわかる。そして、この 3 つのパラメータに関する情報が特殊化対象データとして影響度情報に記述される。

なお、図 1 9 に示したように、汎用コンパイル処理部 1 1 のコンパイルによってサンプルプログラム 3 におけるグローバルデータである `this. offset` や `this. count` がローカル変数に変換されているため、図 2 0 においては、これらのパラメータが他のパラメータ `ch` などと区別なく扱われていることがわかる。

また、図 1 9 のプログラムにおいて太字で記し下線を付した部分が、パラメータ `fromIndex`、`this. offset` 及び `this. count` の位置である。

【 0 0 7 1 】

次に、特殊化データ選択部 1 3 により、パラメータ `fromIndex`、`this. offset` 及び `this. count` に関して、取り得る値ごとに出現頻度の統計が取られる。そして、得られた統計情報に基づいて特殊化実行データが決定される。具体的な統計情報の提示は省略するが、ここでは、`fromIndex` は定数値 0、`this. offset` は定

数値 0、this. count は定数値 4 を取る場合がそれぞれ最も出現率が高く、この場合の出現率と影響度値との乗算値はガードコードの挿入による遅延分の 1 よりも大きいものとする。したがって、パラメータ fromIndex、this. offset 及び this. count は特殊化実行データとされる。

【0072】

以上の処理の後、特殊化コンパイル処理部 14 により、図 19 に示した通常のコンパイル後のサンプルプログラム 3 における fromIndex が値 0 に、this. offset が値 0 に、this. count が値 4 にそれぞれ特殊化されてコンパイルされる。

図 21 は、図 19 のプログラムに対して、かかる特殊化が行われた状態のソースプログラムを示す図である。

【0073】

【発明の効果】

以上説明したように、本発明によれば、影響度解析処理及び統計処理を経ることにより、コンパイルに要する時間を短縮し、かつコンパイル時のメモリ消費量を抑えることができる。

また、本発明によれば、メソッドコールを行う場所で呼び出しメソッドが特定できない場合であっても特殊化を実行可能とすることができる。

【図面の簡単な説明】

【図 1】 本実施の形態におけるコンパイラの全体構成を説明する図である。

【図 2】 本実施の形態によるコンパイラの全体的な処理の流れを説明するフローチャートある。

【図 3】 本実施の形態における特殊化のタイプと特殊化を行う命令との対応関係を示す図表である。

【図 4】 本実施の形態のパラメータの特殊化における影響度解析処理のアルゴリズムを説明する図である。

【図 5】 本実施の形態のパラメータの特殊化における特殊化の効果を見積もる処理のアルゴリズムを説明する図である。

【図 6】 本実施の形態のパラメータの特殊化における特殊化データ選択部

による統計処理のアルゴリズムを説明する図である。

【図 7】 本実施の形態のパラメータの特殊化における特殊化実行データを選択する処理のアルゴリズムを説明する図である。

【図 8】 本実施の形態のパラメータの特殊化における一適用例に用いるサンプルプログラムを示す図である。

【図 9】 図 8 に示したサンプルプログラムに対する影響度解析処理の結果を示す図である。

【図 1 0】 図 8 に示したサンプルプログラムに対する統計情報の例を示す図である。

【図 1 1】 図 8 のサンプルプログラムに対して特殊化が行われた状態のソースプログラムを示す図である。

【図 1 2】 本実施の形態のパラメータの特殊化における他の適用例に用いるサンプルプログラムを示す図である。

【図 1 3】 図 1 2 のサンプルプログラムに対して通常のコンパイルにおける最適化が行われた状態を示す図である。

【図 1 4】 図 1 2 に示したサンプルプログラムに対する影響度解析処理の結果を示す図である。

【図 1 5】 図 1 2 のサンプルプログラムに対して特殊化が行われた状態のソースプログラムを示す図である。

【図 1 6】 図 1 2 のサンプルプログラムに対してさらに他の条件で特殊化が行われた状態のソースプログラムを示す図である。

【図 1 7】 本実施の形態のグローバルデータに対する特殊化における影響度解析処理のアルゴリズムを説明する図である。

【図 1 8】 本実施の形態のグローバルデータに対する特殊化における一適用例に用いるサンプルプログラムを示す図である。

【図 1 9】 図 1 8 のサンプルプログラムに対して通常のコンパイルにおける最適化が行われた状態を示す図である。

【図 2 0】 図 1 9 に示したサンプルプログラムのコンパイル結果に対する影響度解析処理の結果を示す図である。

【図 2 1】 図 1 9 のプログラムに対して特殊化が行われた状態のソースプログラムを示す図である。

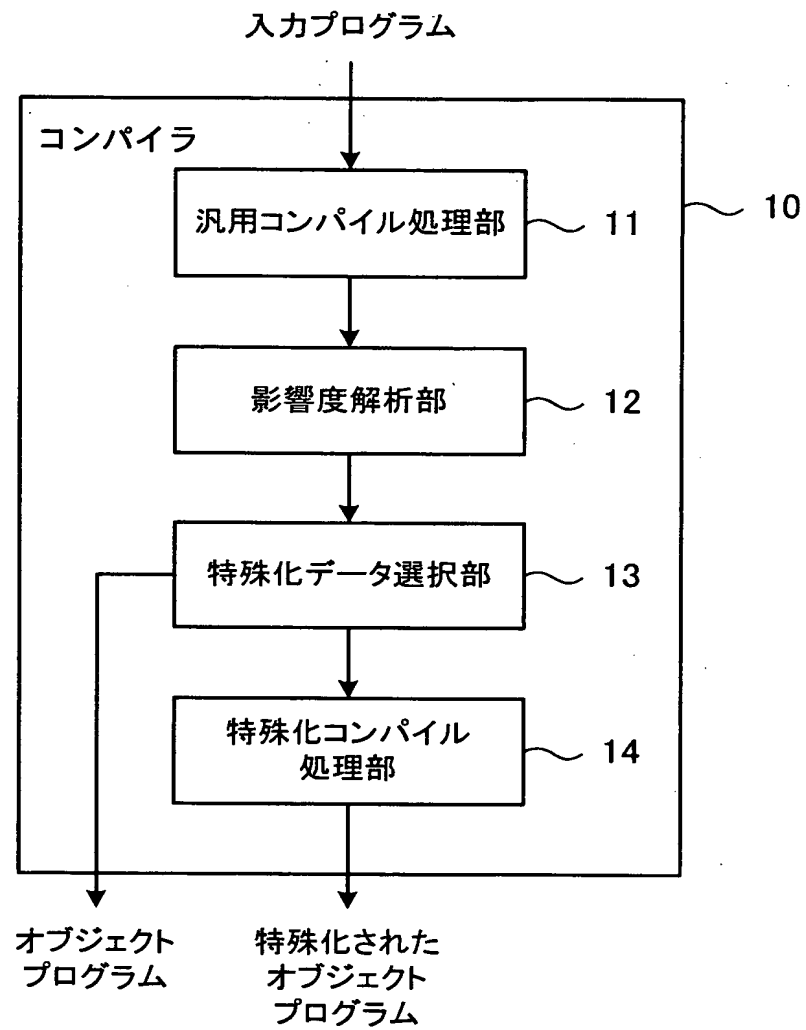
【符号の説明】

1 0 …コンパイラ、1 1 …汎用コンパイル処理部、1 2 …影響度解析部、1 3 …特殊化データ選択部、1 4 …特殊化コンパイル処理部

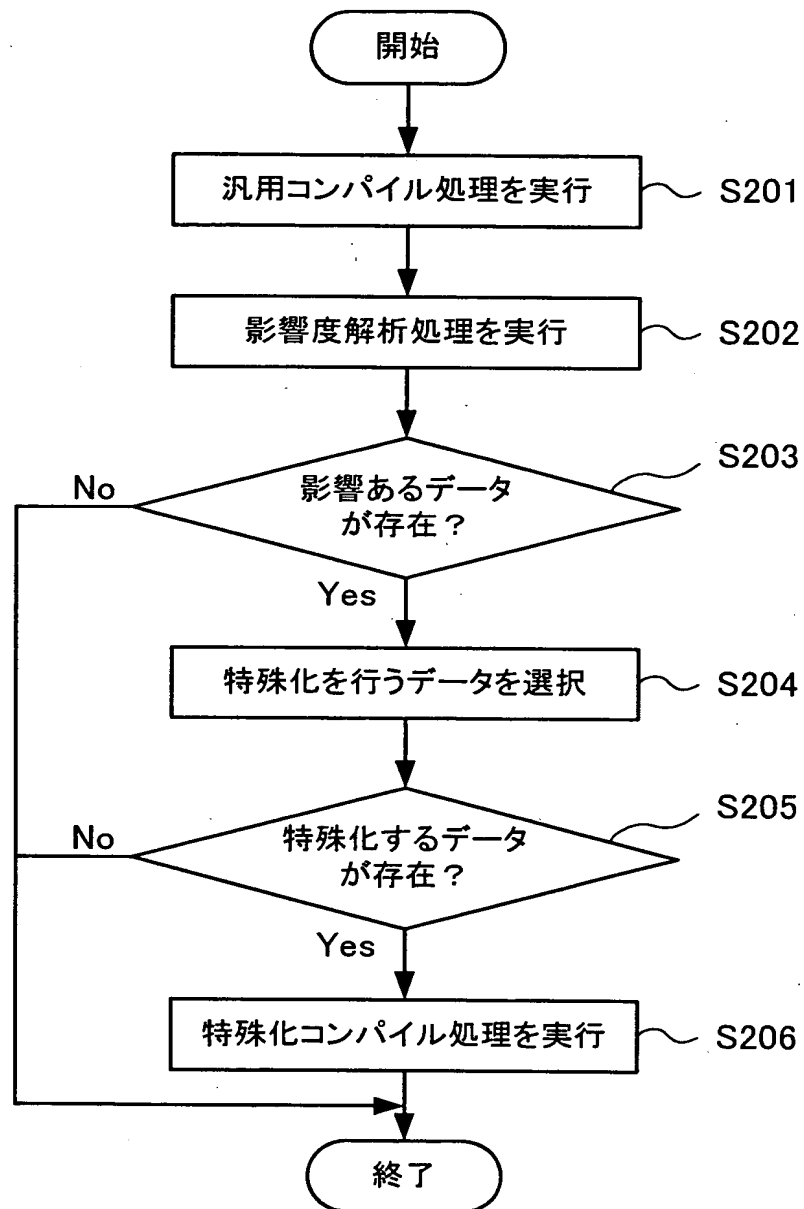
【書類名】

図面

【図 1】



【図 2】



【図 3】

タイプ	命令
定数	四則演算 オブジェクト以外の変数に対する比較命令 (switch命令も含む)
Nullではない	変数がNullかどうかを明示的にチェックする命令全般
クラスの特定	INSTANCEOF CHECKCAST 仮想的なメソッド呼び出し命令全般
配列以外のクラスである	オブジェクトが配列かどうかをチェックする命令全般. 具体的には INSTANCEOF CHECKCAST INVOKEVIRTUALOBJECT_QUICK

【図 4】

```
Effective =  $\phi$ ;  
for (each  $p \in$  全てのパラメータ) {  
  for (each type  $\in$  ALLTYPE) {  
    Impact[p][type] = 0;  
  }  
  Estimate(impact, p, p,  $\phi$ );  
  for (each type  $\in$  ALLTYPE) {  
    if (Impact[p][type] > しきい値A) /* しきい値が1の時ガードコードと同等 */  
      Effective U= { p, type };  
  }  
}
```

【図 5】

```

Estimate(impact, v, p, type)
{
    for (each l ∈ vを使っている命令) { /* DU-chainを使う */
        if (l上のvの定義はただ一つ) { /* UD-chainを使う */
            tmp_type = ALLTYPEの中どの特殊化を行えば, v によって命令lが最適化されるか; - (1)
            if (tmp_type != φ) { /* v によって l が最適化されるならば */
                if (type == φ) {
                    type = tmp_type;
                }
                Impact[p][type] += typeに対するガードコードのコストを1としたときの, 命令lが最
                適化されたときの影響度; /* 例えば, 2ならばガードコードの2倍・0.5ならばガード
                コードの半分の効果がある */
                if (特殊化によりlが定数への代入命令になる) {
                    Estimate(impact, lの結果が入るローカル変数, p, type);
                }
            }
        }
    }
}

```

【図 6】

```
if (Effective ==  $\phi$ ) パラメータの特殊化の処理を中断
for (each { p, type }  $\in$  Effective) {
    パラメータ p について特殊化の方法 type の条件で統計を取る.
}
```


【図 7】

```
Specialize =  $\phi$  ;  
for (each { p, type }  $\in$  Effective) {  
    odds = { p, type } に関する統計の中で, もっとも優秀だった確率;  
    val = oddsに対応する { p, type } の値;  
    if (Impact[p][type] * odds > しきい値 B) { /* しきい値が 1 の時ガードコードと同等 */  
        Specialize U= { p, type, val };  
    }  
}  
if (Specialize ==  $\phi$ ) パラメータの特殊化の処理を中断
```

【図 8】

```
public void getChars(int srcBegin, int srcEnd, char dst[], int dstBegin) {  
    if (srcBegin < 0) {  
        throw new StringIndexOutOfBoundsException(srcBegin);  
    }  
    if (srcEnd > this.count) {  
        throw new StringIndexOutOfBoundsException(srcEnd);  
    }  
    if (srcBegin > srcEnd) {  
        throw new StringIndexOutOfBoundsException(srcEnd - srcBegin);  
    }  
    System.arraycopy(this.value, this.offset + srcBegin, dst, dstBegin,  
        srcEnd - srcBegin);  
}
```

【図9】

(A)

影響度解析が終わったときのImpact[p][type]

p	type	Impact[p][type]
srcBegin	定数	2.25
	それ以外	0
srcEnd	定数	1.25
	それ以外	0
dst	全て	0
dstBegin	全て	0

(B)

```

public void getChars(int srcBegin, int srcEnd, char dst[], int dstBegin) {
    if (srcBegin < 0) {
        throw new StringIndexOutOfBoundsException(srcBegin);
    }
    if (srcEnd > this.count) {
        throw new StringIndexOutOfBoundsException(srcEnd);
    }
    if (srcBegin > srcEnd) {
        throw new StringIndexOutOfBoundsException(srcEnd - srcBegin);
    }
    System.arraycopy(this.value, this.offset + srcBegin, dst, dstBegin,
                     srcEnd - srcBegin);
}

```

【図 1 0】

SPECjvm98のJackベンチマークを使った場合の統計

srcBegin	値 0	確率 100%
srcEnd	値 1	確率 68%
	値 0	確率 15%
	値 2	確率 4%
	値 3	確率 3%
	値 4	確率 3%
	(以下略)	

【図 1 1】

```

public void getChars(int srcBegin, int srcEnd, char dst[], int dstBegin) {
    if (srcEnd > this.count) {
        throw new StringIndexOutOfBoundsException(srcEnd);
    }
    if (0 > srcEnd) {
        throw new StringIndexOutOfBoundsException(srcEnd);
    }
    System.arraycopy(this.value, this.offset, dst, dstBegin, srcEnd);
}

```

【図 1 2】

```

boolean dispatchEvent (AWTEvent e) {
    boolean ret = false;
    if ((e instanceof MouseEvent) &&
        ((eventMask & MOUSE_MASK) != 0)) {
        MouseEvent me = (MouseEvent) e;
        ret = processMouseEvent (me);
    } else if (e instanceof FocusEvent) {
        FocusEvent fe = (FocusEvent) e;
        ret = processFocusEvent (fe);
    } else if (e instanceof KeyEvent) {
        KeyEvent ke = (KeyEvent) e;
        ret = processKeyEvent (ke);
    }
    return ret;
}

```

【図 1 3】

```

boolean dispatchEvent (AWTEvent e) {
    boolean ret = false;
    if ((e instanceof MouseEvent) &&
        ((eventMask & MOUSE_MASK) != 0)) {
        MouseEvent me = e;      /* checkcastが除去された */
        ret = processMouseEvent (me);
    } else if ((e instanceof FocusEvent)) {
        FocusEvent fe = e;      /* checkcastが除去された */
        ret = processFocusEvent (fe);
    } else if ((e instanceof KeyEvent)) {
        KeyEvent ke = e; /* checkcastが除去された */
        ret = processKeyEvent (ke);
    }
    return ret;
}

```

【図 1 4】

影響度解析が終わったときの Impact [p] [type]

p	type	Impact [p] [type]
e	クラスの特定	3. 94
	それ以外	0

【図 1 5】

```

boolean dispatchEvent (AWTEvent e) {
    boolean ret = false;
    if (e != null) {
        KeyEvent ke = e;
        ret = processKeyEvent (ke);
    }
    return ret;
}

```

【図 1 6】

```

boolean dispatchEvent (AWTEvent e) {
    boolean ret;
    KeyEvent ke = e;
    ret = processKeyEvent (ke);
    return ret;
}

```

【図 17】

```
Effective =  $\phi$ ;  
ALLTYPE = 全ての特殊化するタイプ  
for (each  $p \in \text{DownSafety}[\text{メンソッドの先頭}]$ ) (  
    lvar =  $p$ の結果が格納されたローカル変数;  
    for (each  $\text{type} \in \text{ALLTYPE}$ ) (  
        Impact[ $p$ ][ $\text{type}$ ] = 0;  
    )  
    Estimate(Impact, lvar, lvar,  $\phi$ );  
    for (each  $\text{type} \in \text{ALLTYPE}$ ) (  
        if (Impact[ $p$ ][ $\text{type}$ ] > しきい値A) (/* しきい値が1の時ガードコードと同等 */  
            Effective  $\cup = \{p, \text{type}\}$ ;  
        )  
    )  
)
```

【図 18】

```

public int indexOf(int ch, int fromIndex) {
    int max = this.offset + this.count;
    char v[] = this.value;

    if (fromIndex < 0) {
        fromIndex = 0;
    } else if (fromIndex >= this.count) {
        return -1;
    }
    for (int i = this.offset + fromIndex ; i < max ; i++) {
        if (v[i] == ch) {
            return i - this.offset;
        }
    }
    return -1;
}

```

【図 19】

```

public int indexOf(int ch, int fromIndex) {
    int offset = this.offset;
    int count = this.count;
    int max = offset + count;
    char v[] = this.value;

    if (fromIndex < 0) {
        fromIndex = 0;
    } else if (fromIndex >= count) {
        return -1;
    }
    for (int i = offset + fromIndex ; i < max ; i++) { -- (1)
        if (v[i] == ch) {
            return i - offset;
        }
    }
    return -1;
}

```


【図 2 0】

影響度解析が終わったときのImpact[p][type]

p	type	Impact[p][type]
ch	全て	0
fromIndex	定数	1.75
	それ以外	0
this.offset	定数	5.75
	それ以外	0
this.count	定数	5.75
	それ以外	0
this.value	全て	0

【図 2 1】

```

public int indexOf(int ch, int
fromIndex) {
    char v[] = this.value;

    if (v[0] == ch) return 0;
    if (v[1] == ch) return 1;
    if (v[2] == ch) return 2;
    if (v[3] == ch) return 3;
    return -1;
}

```

【書類名】 要約書

【要約】

【課題】 特殊化による最適化を含むコンパイルにおいて、コンパイルに要する時間を短縮し、コンパイル時のメモリ消費量を抑える。

【解決手段】 コンパイラ 1 0 において、コンパイルの対象であるプログラムにおける所定の命令のパラメータを特定の状態に固定することによりこのプログラムの実行速度をどれだけ向上させることができるかを解析する影響度解析部 1 2 と、このプログラムを実行した場合に、この影響度解析部 1 2 にて解析された命令のパラメータが取り得る状態ごとの出現頻度の統計を取り、得られた統計情報に基づいてこの命令のパラメータをどの状態に固定するかを決定する特殊化データ選択部 1 3 と、この影響度解析部 1 2 及び特殊化データ選択部 1 3 による処理結果に基づいて、このプログラム中に、所定の命令のパラメータを特定の状態に固定した特殊化されたパスを生成する特殊化コンパイル処理部 1 4 とを備える。

【選択図】 図 1

認定・付加情報

特許出願の番号	特願 2001-055996
受付番号	50100287162
書類名	特許願
担当官	風戸 勝利 9083
作成日	平成 13 年 4 月 20 日

<認定情報・付加情報>

【特許出願人】

【識別番号】	390009531
【住所又は居所】	アメリカ合衆国 10504、ニューヨーク州 アーモンク (番地なし)
【氏名又は名称】	インターナショナル・ビジネス・マシーンズ・コーポレーション

【代理人】

【識別番号】	100086243
【住所又は居所】	神奈川県大和市下鶴間 1623 番地 14 日本アイ・ビー・エム株式会社 大和事業所内
【氏名又は名称】	坂口 博

【代理人】

【識別番号】	100091568
【住所又は居所】	神奈川県大和市下鶴間 1623 番地 14 日本アイ・ビー・エム株式会社 大和事業所内
【氏名又は名称】	市位 嘉宏

【代理人】

【識別番号】	100106699
【住所又は居所】	神奈川県大和市下鶴間 1623 番 14 日本アイ・ビー・エム株式会社大和事業所内

【氏名又は名称】	渡部 弘道
----------	-------

【復代理人】

【識別番号】	100104880
【住所又は居所】	東京都港区赤坂 5-4-11 山口建設第 2 ビル 6 F セリオ国際特許事務所

【氏名又は名称】	古部 次郎
----------	-------

【選任した復代理人】

【識別番号】	100100077
--------	-----------

次頁有

認定・付加情報（続き）

【住所又は居所】 東京都港区赤坂5-4-11 山口建設第2ビル
6F セリオ国際特許事務所
【氏名又は名称】 大場 充

出 願 人 履 歴 情 報

識別番号 [390009531]

1. 変更年月日 2000年 5月16日

[変更理由] 名称変更

住 所 アメリカ合衆国10504、ニューヨーク州 アーモンク (番地なし)

氏 名 インターナショナル・ビジネス・マシーンズ・コーポレーション